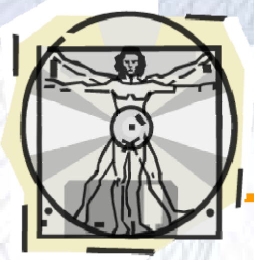
UNIVERSITY OF CALGARY

SENG 697
Agent-based Software Engineering

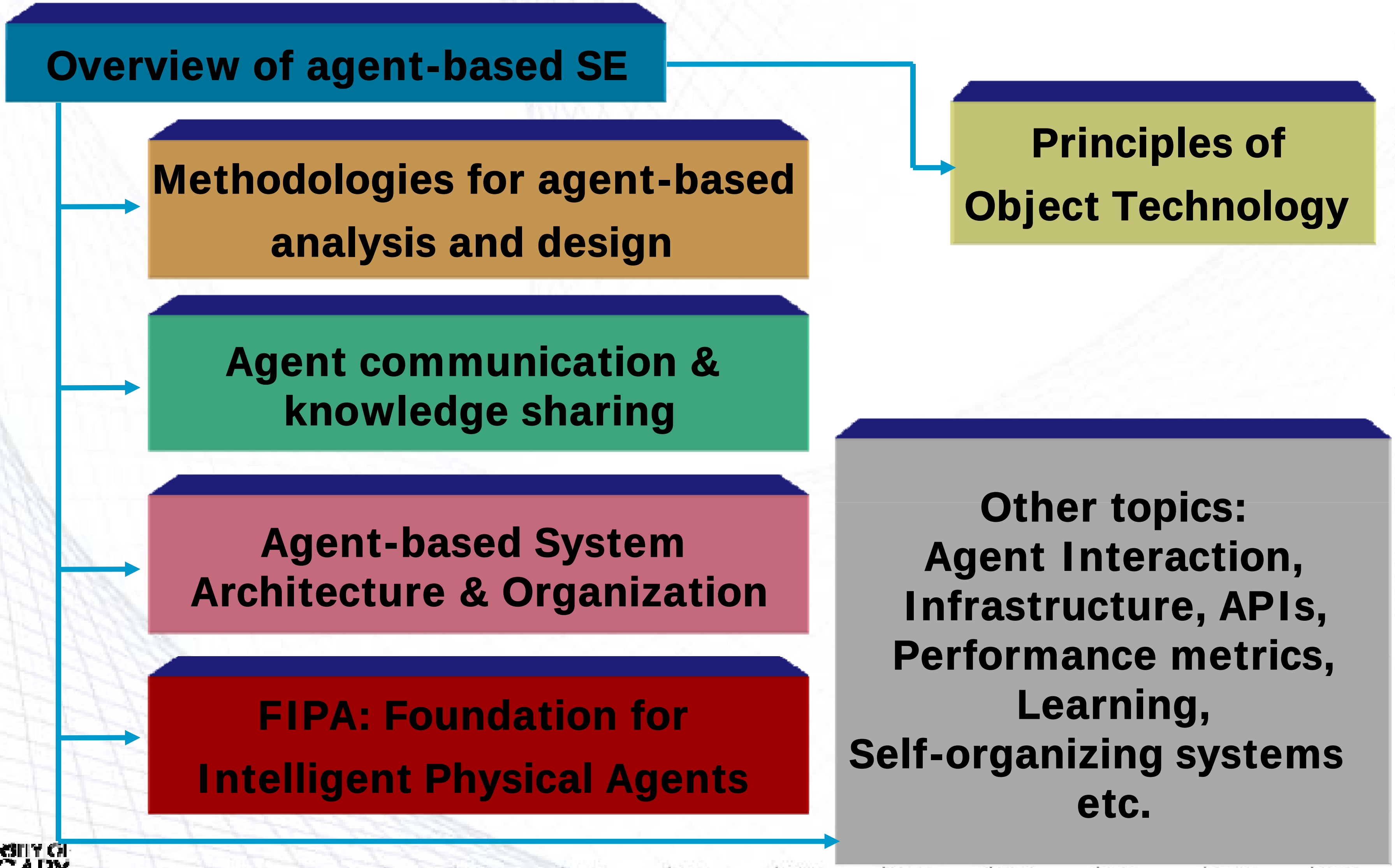
Behrouz Far
Schulich School of Engineering, University of Calgary
far@ucalgary.ca
http://www.enel.ucalgary.ca/People/far/



Contents

- Introduction
- Software vs. software agent
- Software agent models
- Agent decision making process
- Conclusion

Course Curriculum



Overview of agent-based SE

Methodologies for agent-based analysis and design

Agent communication & knowledge sharing

Agent-based System Architecture & Organization

FIPA: Foundation for Intelligent Physical Agents

Principles of Object Technology

Other topics:
Agent Interaction,
Infrastructure, APIs,
Performance metrics,
Learning,
Self-organizing systems
etc.

Realities: High Risk

- Software development is a very high risk task (may be!)
- About 20% of the software projects are canceled
- About 84% of software projects are incomplete when released (need patches, service packs, etc.)
- Almost all of the software projects costs exceed initial estimations (cost overrun)
- **Question:** What can be done to improve **software product** and **software development process**?

Software Product: Trends …

- **Past:** centralized computing system; monolithic programming model
- **Now:** distributed computing system; heterogeneous, scalable, open and distributed programming model

Nowadays, an increasing number of software projects are revised, restructured and reconstructed in terms of *software agents*: MAS

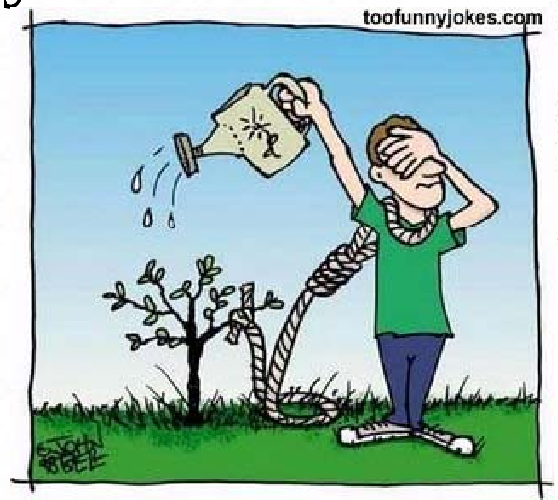
© 1999 Randy Glasbergen. www.glasbergen.com

"It's the latest innovation in office safety. When your computer crashes, an air bag is activated so you won't bang your head in frustration."

Software Process: Trends …

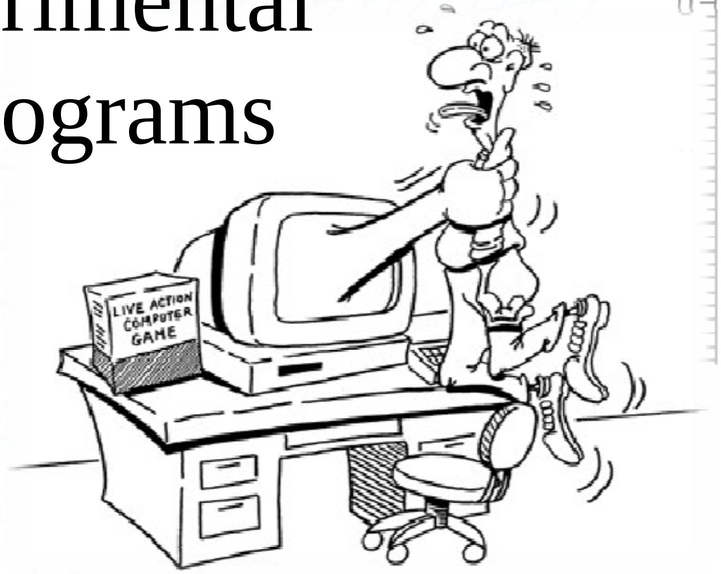
- **Past:** waterfall development process model
- **Now:** incremental, agile, experimental development process model

- **Agent-based software development:** new way of analysis and synthesis of software systems (supposed to be …)



Paradigm Shift …

- May be the “software concept” itself should be revised not only the process and/or the product
- **Software agents:** new experimental embodiment of computer programs

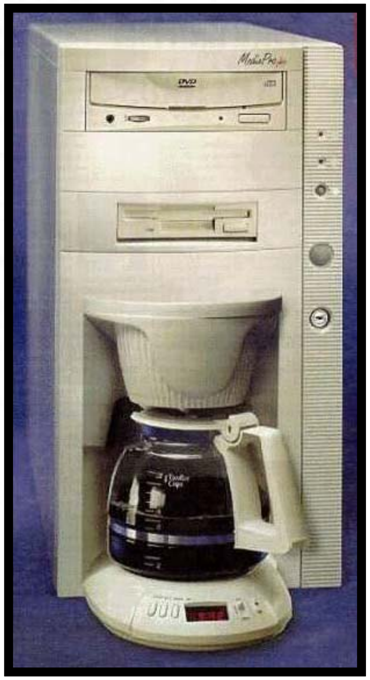


Controversial Definitions /1

- An agent is a **software entity** that functions continuously and autonomously in a particular environment, often inhabited by other agents and processes [Shoham 97]

Possible examples:

A cell phone?
A time controlled sprinkler?
A notification program: e.g., CCTray?



- Other concerns such as weight, knowledge, learning, inference capability, etc. … are equally important

Controversial Definitions /2

An agent is a software entity that might possess some of the following attributes

- **Reactivity** (i.e., selectively sense and act)
- **Autonomy** (i.e., goal directness, proactive & self-started behavior)
- **Collaborative** (i.e., work with other agents and entities to achieve a common goal)
- **Knowledge-level communication ability** (i.e., communicate with other entities in a language like speech-act, higher level than symbol level program to program protocols)
- **Inferential capability** (i.e., act on abstract task spec., using models of self, situation, and/or other agents)
- **Temporal continuity** (i.e., persistence of state and personality)
- **Personality** (i.e., manifesting attributes of a believable agent. Personality is composed of beliefs and behavior based on those beliefs)
- **Adaptability** (i.e., learn and improve with experience)
- **Mobility** (i.e., migrate from one host to another in a self-directed way)

Controversial Definitions /3

An agent is a software entity that exhibits (a subset of) the following characteristics:

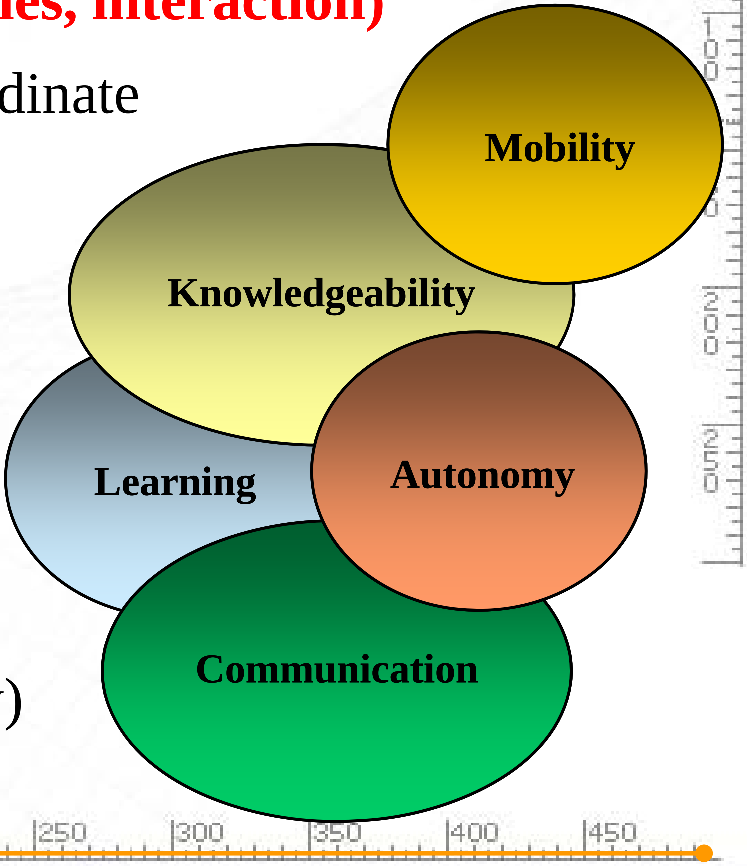
- **Knowledgeability (cognitive capabilities, interaction)**

  Capability to interact (cooperate, coordinate and compete) with the other people, agents and processes

- **Learning**

  Capability to learn autonomously

- **Autonomy** (set own goal, proactive)
- **Communication** (high level language)
- **Mobility** (physical or software mobility)



Controversial Development

- Agent-oriented programming
- Agent-oriented software development
- Agent-oriented design

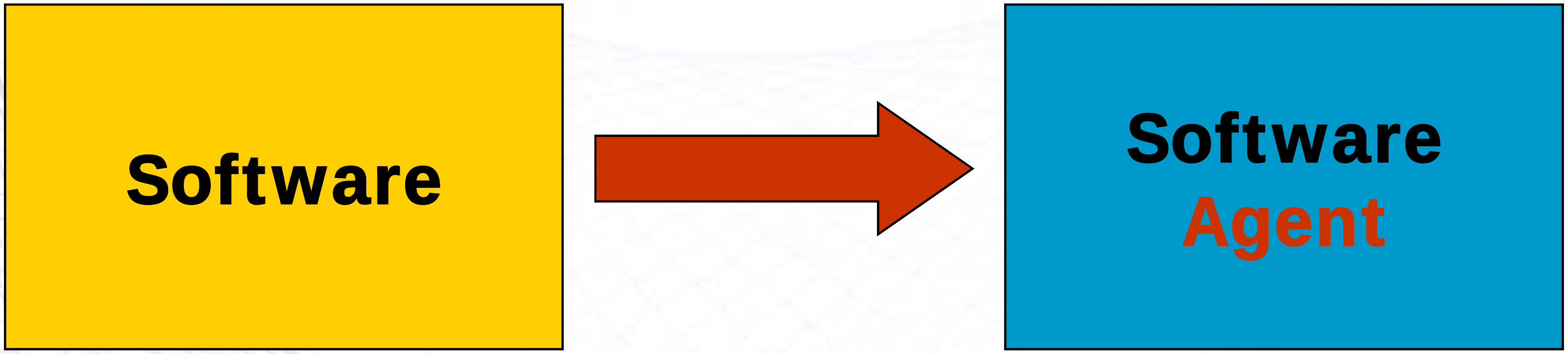
etc.

"I think we've just received the winning resume for that new secretary position."

Agent system development is dominated by informal guidelines, heuristics and inspirations rather than formal principles and well-defined engineering techniques. [Jennings] Good on paper… hard to use in practice

Software vs. Software Agent

- Similarities and differences:
  - **Software** aspect of a software agent
  - **Agent** aspect of a software agent



Software

Software Agent

What Makes them Different?

1. Roles vs. tasks
2. Emergent behaviour
3. Knowledge cycle
4. Interactions vs. intra-actions
5. Symbol level communication
6. Knowledge completeness
7. Indeterminism: Decision making

1. Roles vs. Tasks

Use Cases: How to use bathroom

- We assign "**tasks**" to software

  Example: on-line registration software

  - "What" and "how" are specified in advance (use cases, scenarios, etc.)
  - Changes in requirements are not tolerable

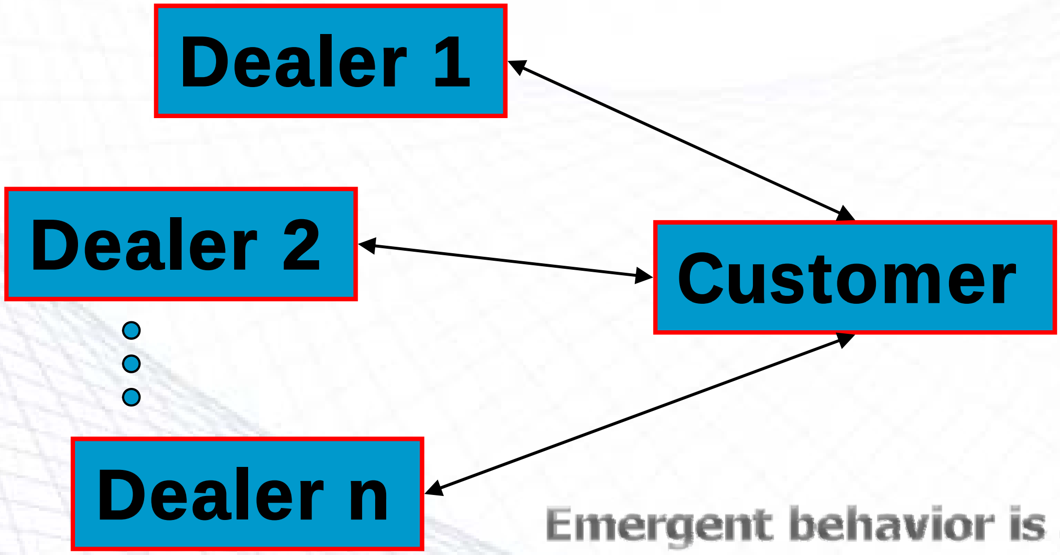
- We assign "**roles**" to software agents

  Example: registration agent; travel agent; secretary

  - "What" is specified in advance. "How" is determined dynamically ← **"how to" library**
  - Changes in requirement can be tolerated

2. Emergent Behaviour

- Agent system behaviour is not "fully predefined". It is the result of dynamic interaction among the participants
- Example: Electronic Commerce System



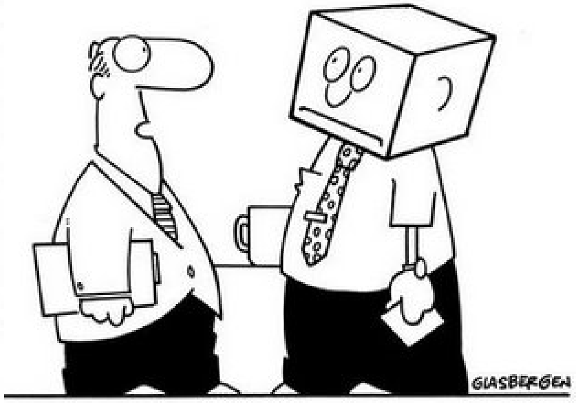
Dealer agents learnt that they can maximize their profit by sharing the invoice price among themselves!

Emergent behavior is appreciated not avoided

3. Knowledge Cycle

- Data ⇒ Information ⇒ Knowledge
  - ***Data*** is a sequence of quantified or quantifiable symbols
  - ***Information*** is about taking data and putting it into a meaningful pattern
  - ***Knowledge*** is the ability to use that information
- ***Intelligence*** ⇒ *Data* ⇒ *Information* ⇒ *Knowledge*
- How intelligence can be gathered?
  - Informants, consultants, experts
  - Yellow pages, Web, external signals, knowledge-bases

Copyright 2005 by Randy Glasbergen. www.glasbergen.com



"Thinking outside of the box is difficult for some people. Keep trying."

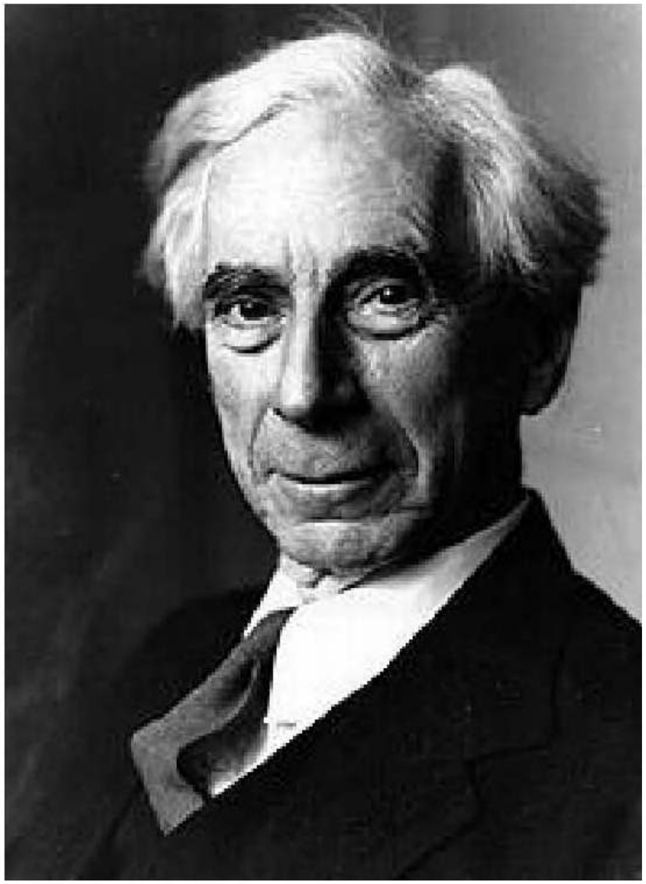
Knowledge Cycle (cont'd)

Theory of Knowledge

(Encyclopaedia Britannica)

Concepts:

Knowledge, belief, words, personality, behaviour, truth, uncertainty, data, inference, probability, etc.



Bertrand Russell

4. Interactions vs. Intra-actions

- In software systems, relationships among subsystems (or components, packages) are:
  - *Inter-actions* among subsystems and
  - *Intra-actions* within a subsystem
- *Interactions* are between the artifact and its outer environment. E.g., giveaway: give the puck to the other team
- *Intra-actions* are the characteristics of the artifact’s inner environment. E.g., pass: pass the puck to a teammate

4. Interactions vs. Intra-actions

- In nearly decomposable systems *intra-actions* are more frequent (typically at least 10 times more) and more predictable than *inter-actions* [Herbert Simon].
- Software systems are nearly decomposable: subsystems can be treated almost independently.
- Although many of the *inter-actions* can be predictable at design time, some simply cannot.

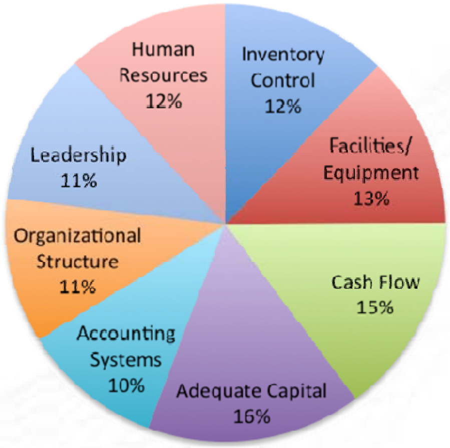
Therefore

- Conventional software engineering approaches can handle well *intra-actions* within or among components.
- Agent-approach is needed to handle *inter-actions* among (clusters of) software components. ← **focus on architecture**

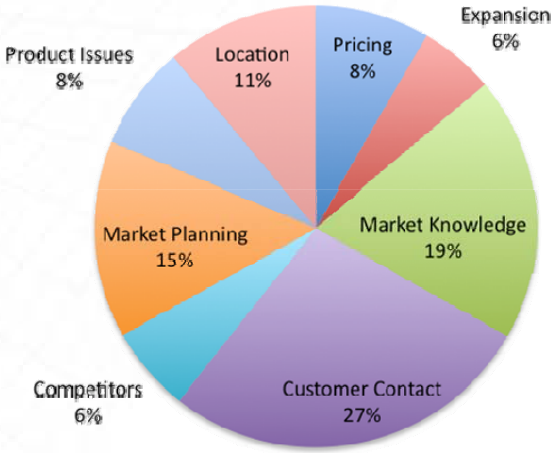
Example: Inter vs. Intra-actions

- Internal and External Problems Experienced by Entrepreneurs

Internal Problems

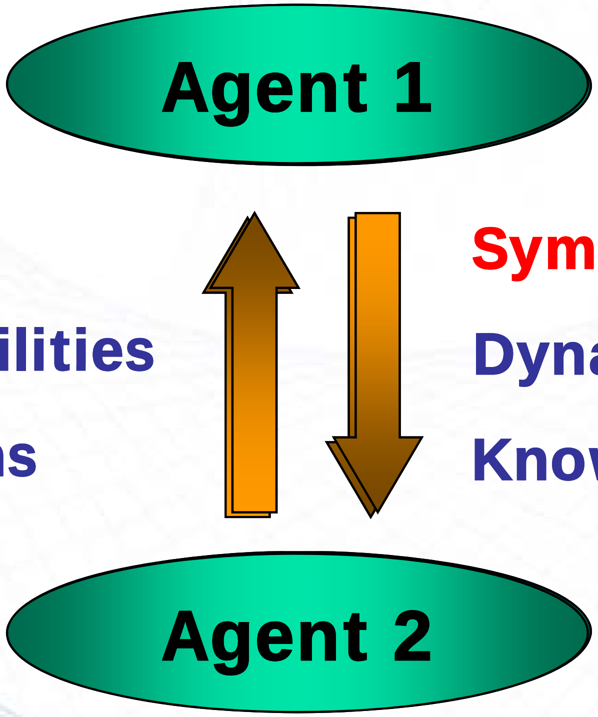


External Problems



5. Symbol Level Communication

- Software: Signal-level communication
- Agents: Symbol-level communication



Signal level:

Assign responsibilities

Invoke operations

Symbol level:

Dynamic Message Passing

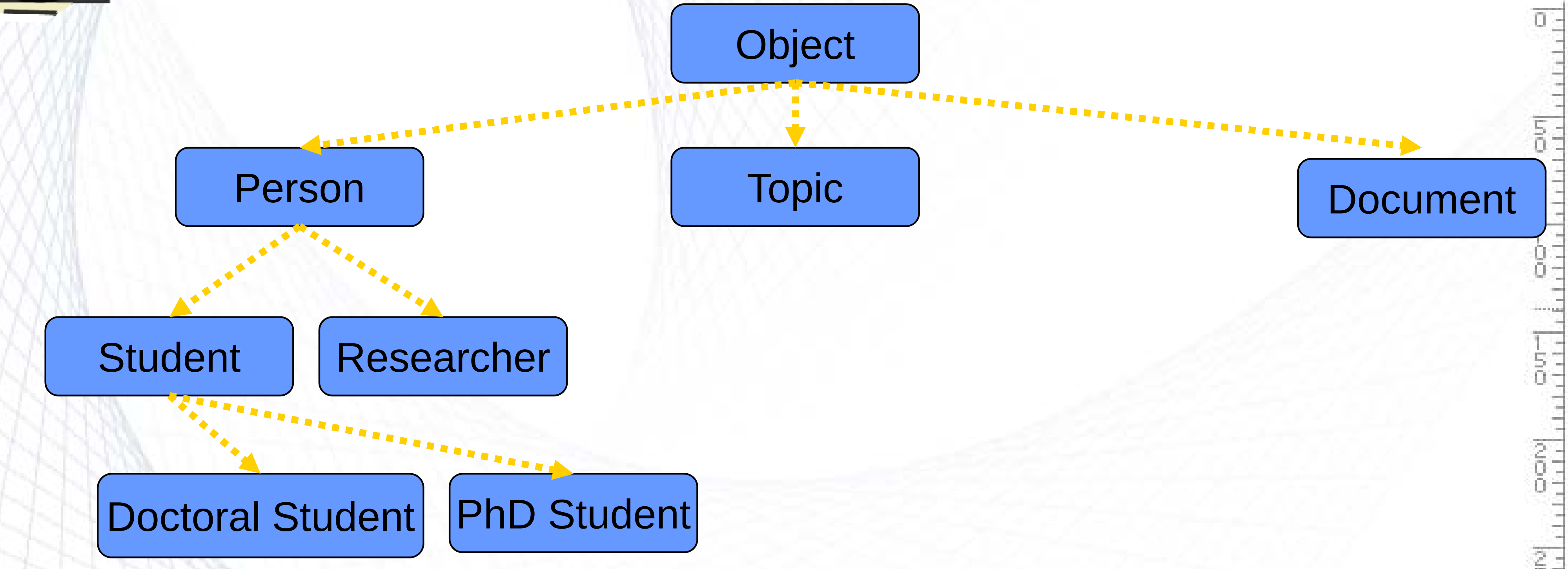
Knowledge sharing

5. Symbol Level Communication

- For a message from one agent to be understood by another, the agents must ascribe the same meaning to the constants used in the message
- Thus, we require an “**ontology**” to map a given constant to some well-understood meaning
- **Dynamic Message Passing:**
  - Agents should first discover whether or not they share a mutual understanding of the domain constants before further message passing

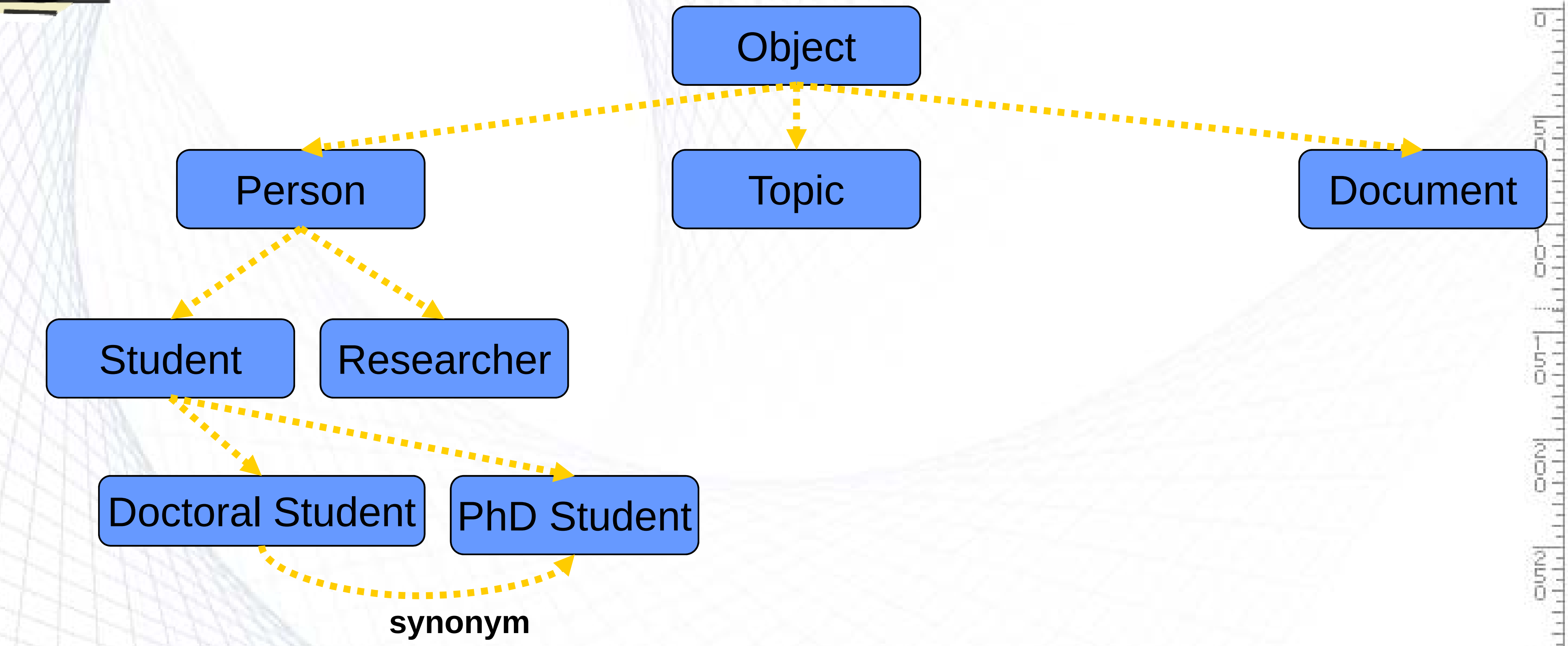
'Information sharing' - you tell me everything about yourself and I'll tell you when to stop.
CHRIS

Taxonomy: What is?



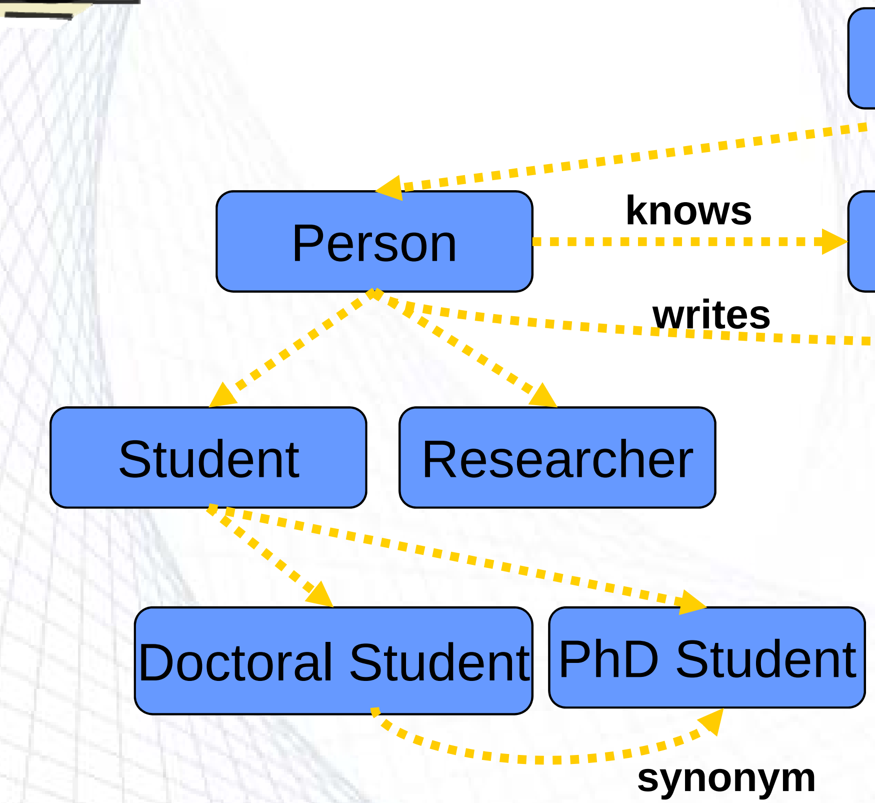
Taxonomy := Segementation, classification and ordering of elements into a classification system according to the relationships among them

Thesaurus: What is?



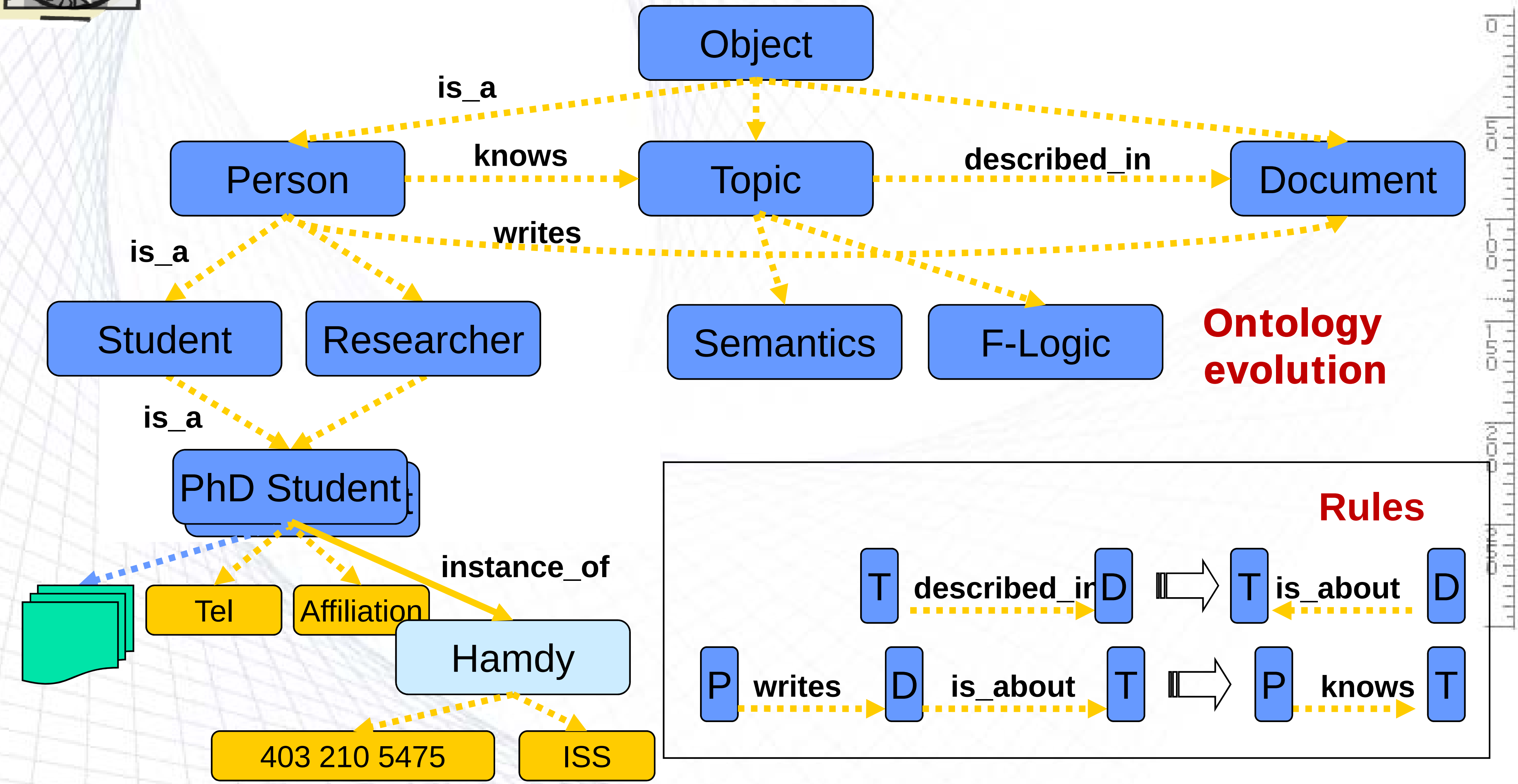
- Terminology for specific domain
- Graph with primitives, 2 fixed relationships (similar, synonym)
- Originated from bibliography

Topic Map: What is?



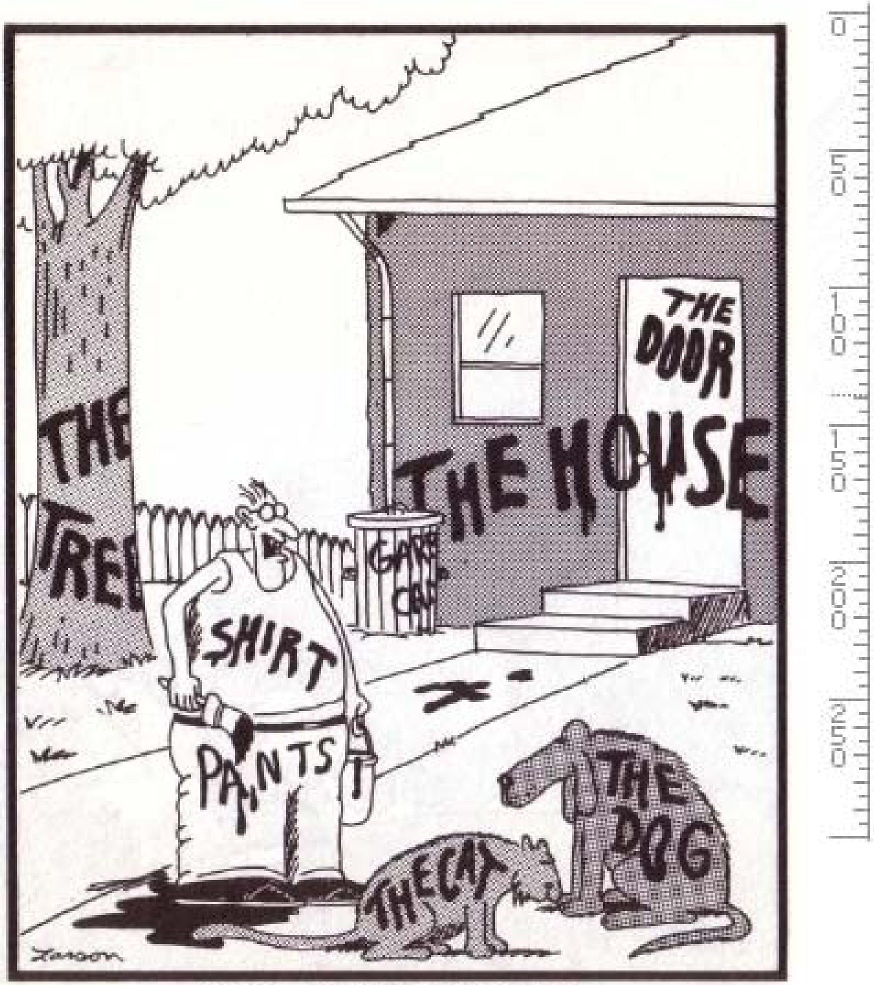
- Topics (nodes), relationships and *occurences* (to documents)
- Typically for navigation- and visualisation

Ontology: What is?



Ontology

- Not only conceptualization but also relations
- May be no good for human but good for machine processing



"Now! ... That should clear up a few things around here!"

6. Knowledge Completeness

- Interacting software systems ***must*** have complete knowledge of each other
- Interacting software agents ***may*** have complete knowledge about the other agents’ goals, strategies (i.e., actions to select from) and utilities (i.e., the pay-offs of actions)
- Interactions based on the complete knowledge assumption are usually classified as ***cooperation*** and ***coordination***
- In many cases knowledge completeness assumption may not hold: ***competition***

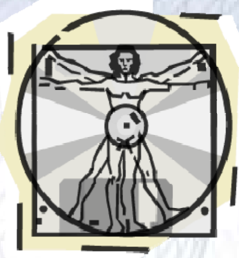
7. Indeterminism: Decision Making

- **Software systems:** Decision points in algorithms are deterministic
- **“Intelligent” software systems:** empowered by *Reasoning* mechanisms
- ***Reasoning*** is based on a single thread of control

  ← has long term consequences
- ***Decision making*** is based on multiple threads of control!

  ← has immediate consequences

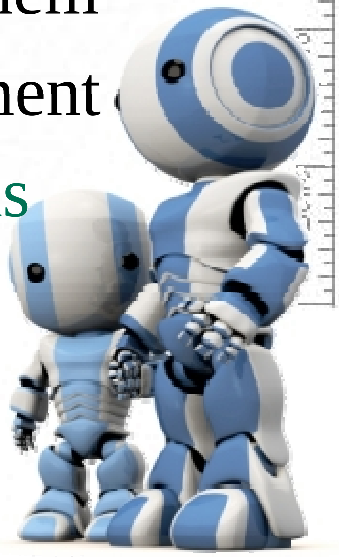
geocities.com/mange1.geo



Therefore: Definition

- Software agents are
  - ***Knowledgeable:*** capable of managing their own set of beliefs (information) and/or desires (goals) and they can decide upon the next operations to execute
  - ***Autonomous:*** we don’t define tasks for them instead we assign roles and responsibilities to them
  - ***Situated:*** defined with respect to their environment
  - ***Interactive:*** capable of filtering communications and managing dialogues

software entities.



Agent System Development

- Analogy:
  - Software agent vs. “movie actor” playing “roles”
  - Use-cases vs. “screen play” or “scenario”
  - Software “designer” and “engineer” vs. “producer” and “director”
- Agent development methodologies:
  - 30+ available
  - 6+ Frequently used

Software Agent Models

1. Agent Models

- There are various ways of modeling and viewing software agents.
- Examples:
  - Computer science models: e.g., indeterministic push-down automaton
  - Software engineering models: based on the object-agent analogy [Jennings and Wooldridge's works]
  - Interaction models: subsystems; software components; or packages

1. Agent – Object Analogy

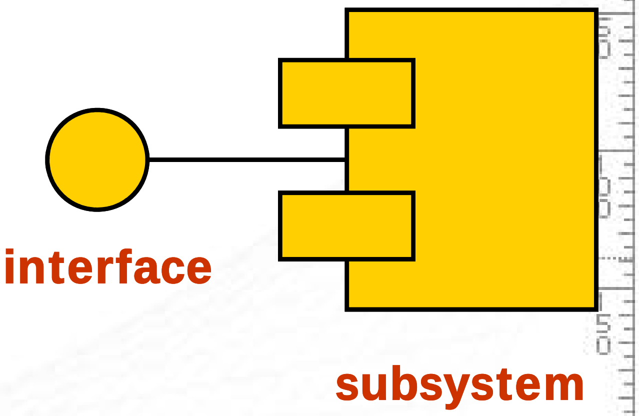
- Object-agent analogy can potentially be misinterpreted and can lead to overselling the agent-based approaches.
- Because *object* is a component level concept and *agent* is an application level one.
- Similarity of *agent-oriented* and *object-oriented* terms should not imply that agent and object are similar entities.

1. Agent – Object Analogy (contd.)

- **Object-oriented software engineering:** engineering a computer program based on objects which are its building block and we use object-oriented methodologies during software analysis and design. ← **building software with objects**
- **Agent-oriented software engineering:** development of a computer program that has properties of an agent and it involves concepts like knowledgeability, autonomy and interactivity. We use agent development methodologies for analysis and design and we use objects to implement them. ← **building software with agent capabilities**

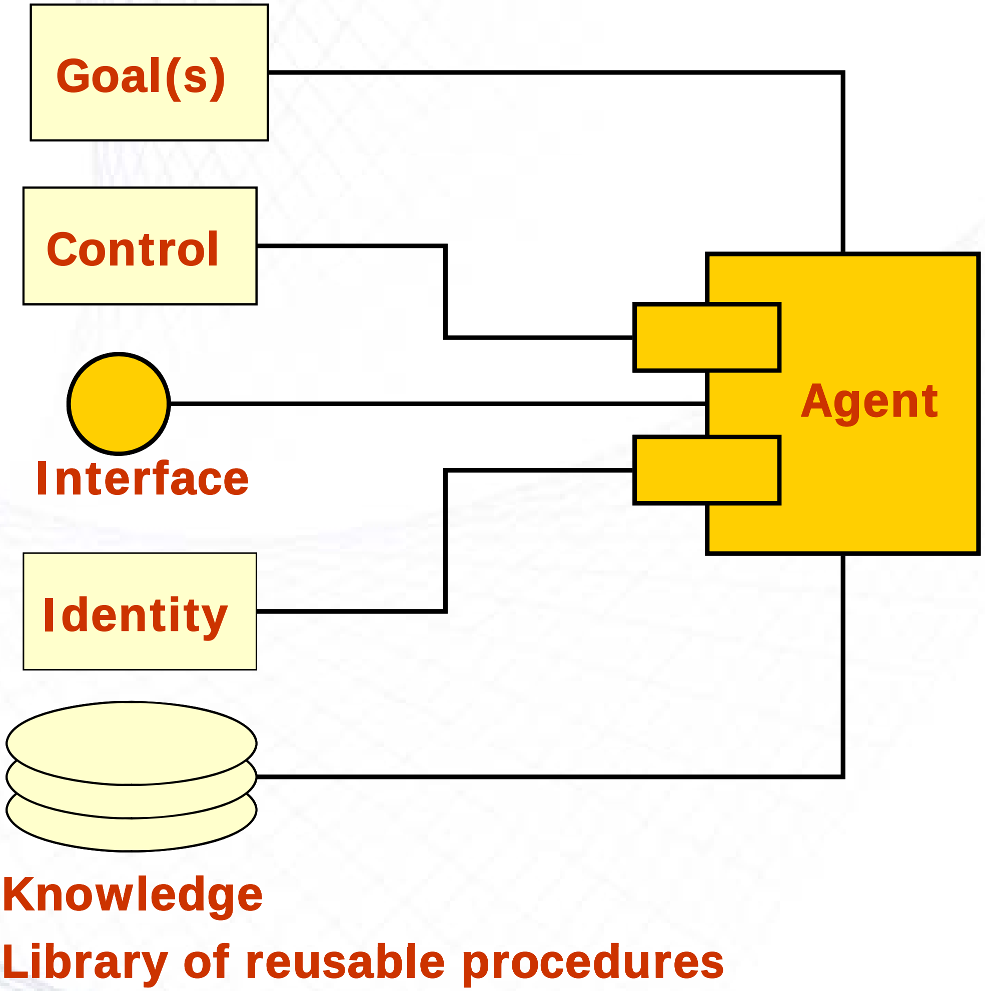
2. Subsystem Analogy

- Modeling agents as *software components*, i.e., a subsystem with complete encapsulation of its behavior that has an attribute, called interface (i.e., what they can do). The component can be accessed through its interface.
- An agents requiring services of another agent may consult directory and naming services (i.e., agent yellow pages) and use the services that the agent offers by adhering to the rules specified in the interface document for that agent.



Experience shows that this limits the scope of the software agents, in the sense that autonomy, proactiveness and interactivity may be compromised.

Enhanced Interaction Model



What I want to do? — Goal(s)

When to do?
What resources I need? — Control

What can I do? — Interface

Who am I?
What is my security clearance? — Identity

How to do? — Knowledge

Library of reusable procedures

Agent

Enhanced Model: Attributes

- Attributes:
- Interface (I) (i.e., what the agent can do?)
- Goal list (G) (i.e., what the agent wants to do?)
- Knowledge (K) (i.e., how to do?)
- Control (C) (when to do?)
- Identity (Id) (whom to contact?)

Enhanced Model: Attributes

- I-G-K-C-Id attributes can be declared ***public***, ***private*** or ***protected***
- ***Public:*** means that the attribute is accessible and readable by all the other agents
- ***Protected:*** means that the attributes are visible only to a certain group of agents
- ***Private:*** indicates that the attributes are not visible externally
- Combination of the attributes and their states lead to various interaction scenarios
- Each interaction scenario has certain properties and satisfying those properties requires implementation of certain reasoning and decision making mechanisms

Interaction Scenarios

- Using the enhanced model, each agent can decide upon the next task to accomplish using the current I-G-K-C-Id list: interfaces, goals, knowledge, identity and thread of control of *self* and the *other* agents with whom interacting.

Cooperation & Coordination

Cooperation & Coordination Scenario		
Attribute	***Visibility***	***Requirements***
Goal (G)	Public	Necessary requirement
Knowledge (K)	Public	
Interface (I)	Public	Full automation and bilateral service
	Protected or Private	Unilateral service only (publish/subscribe)
Control (C)	Public	Sync and/or concurrency
	Protected or Private	No sync and/or concurrency
Identity (Id)	Public	Full communication and bilateral service
	Protected or Private	Unilateral service only

Requirements:

- Knowledge sharing and semantics level message passing methods are needed.
- Additional decision making and/or reasoning methods are not needed.
- Uncertainty management techniques are not needed.

Competition

Loose Competition Scenario		
Attribute	***Visibility***	***Requirements***
Goal (G)	Public	Necessary requirement
Knowledge (K)	Private	
Interface (I)	Public	Comparison of services
	Protected or Private	Encapsulate own
Control (C)	Public	Strategic or dynamic game
	Protected or Private	Strategic game only
Identity (Id)	Public	Role changing is impossible
	Protected or Private	Role changing possible

Requirements:

- Ability to gather and interpret signals indirectly.
- Additional decision making and/or reasoning methods, based on certainty and uncertainty, such as game and utility theory are needed.

Case Studies

- When an agent interacts with the other agents, it is assumed that it has full knowledge of the state of its own I-G-K-C-Id attributes. However, the status for the other agent may be:
  - ***Case 1:*** fully known
  - ***Case 2:*** unknown but the probability distribution over the triple states is known
  - ***Case 3:*** unknown and the probability distribution over the triple states are unknown

Case 1

- Agent_1 knows exactly what the state of the attributes of the Agent_2 is.
- In this case, decision making for Agent_1 is straightforward: Agent_1 can usually select its strategy based on maximum expected utility.

Case 2

- Agent_1 does not know the exact states of the attributes of Agent_2 but it has a probability distribution over the states of Agent_2.
- In this case, Agent_1 treats the known probability distribution as its “belief” and selects the strategy which maximizes its expected utility but broaden the notion of value to reflect the agent’s attitude towards risk.

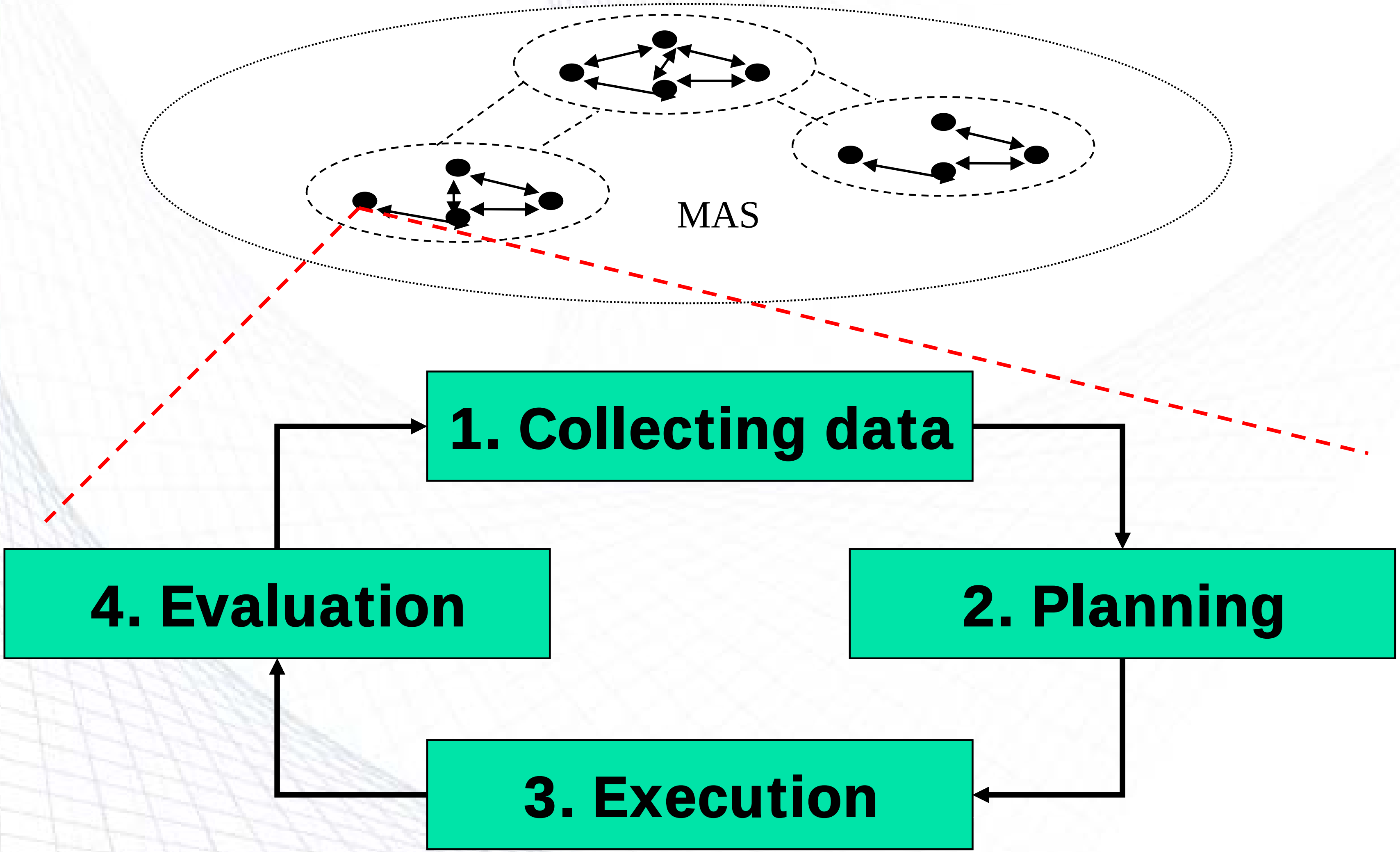
Case 3

- Agent_1 doesn’t know anything about the state of the attributes of Agent_2 except for that it is public or private or protected.
- According to cognitive psychology, when probability distribution is not known, people evaluate belief based on *degree of comfort* (i.e., selecting the alternative that needs the least effort) or *degree of optimism* (i.e., selecting the alternative that we think is the most fit).
- Agent_1 needs a belief assignment method which reflects agent’s degree of optimism.

UNIVERSITY OF CALGARY

Indeterminism: Agent Decision Making Process

Decision Making Process



1. Collecting data

2. Planning

3. Execution

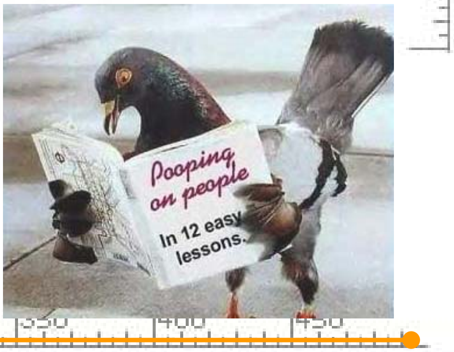
4. Evaluation

Decision Making Process (contd.)

- Planning step:
  1. Organizing and interpreting data/information.
  2. Building the representation model based on different options (i.e. class of games in game theory, utility theory or other uncertainty management theories).
  3. Calculating the expected utilities of possible alternative solutions associated with each option, selecting the *best* one, and using it to select a proper action.

Decision Making Under Certainty

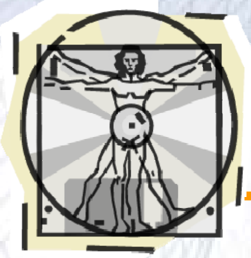
- Use Game Theory
- Agents may take their action simultaneously or in sequence: either strategic game or extensive game can be used.
- *Strategic game:* the agent chooses a strategy that optimizes its utility **(Nash Equilibrium)**
- *Extensive game:* the agent chooses a strategy according to the **Subgame Perfect Equilibrium**



Decision Making Under Uncertainty (without Risk)

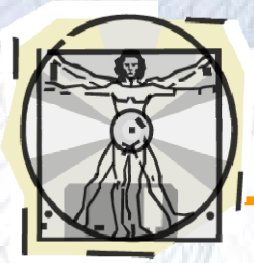
- Use Utility Theory
- Decisions (selecting an action) depend on:
  - What the agent knows (beliefs about various states of the world)
  - What the agent wants (desires; goals)

- We can represent beliefs by ***probabilities***
- We can represent desires by ***utilities*** (*i.e.* benefit or value)



Decision Making Under Uncertainty (without Risk)

- Maximize expected value of the act
  - When considering an act, take into account both the *probability* and the *utility* of each possible consequence.
  - Multiply the probability of each consequence by its utility and then adding them all up.
  - Select the strategy with highest expected value
- This is certainly a “rational” choice but may not be always the “best” one!
- It depends on how we define “utility”
- ***It may not necessarily be irrational to act even when the expected value of the act is negative.***



What is Risk?

- *Risks* risk is a combination of an abnormal event or failure and the consequences of that event or failure to a system’s operators, users, or environment. A risk can range from catastrophic to negligible. Risks are also categorized according to the likelihood of occurrence. [David Gluch]

Severity	**Likelihood of Occurrence**			
	Probable	**Occasional**	**Remote**	**Improbable**
Catastrophic	High	High	High-Medium	Medium
Critical	High	High-Medium	Medium	Medium-Low
Marginal	High-Medium	Medium	Medium-Low	Low
Negligible	Medium	Medium-Low	Low	Low

Decision Making Under Uncertainty (with Risk)

- What if the expected values of two strategies are the same?
- Example:
  - Consider a case where you toss a fair coin and win $1,000,000 if it comes up heads and lose $1,000,000 if it comes up tails.
  - The expected value of “win” and “lose” strategies are equal.
  - The expected value rule says that each choice is of equal value and you should not prefer one to the other.
- Is this right?

Decision Making Under Uncertainty (with Risk)

- Decision depends on our ***attitude towards risk***
- May be you think, then losing $1,000,000 is far worse than gaining $1,000,000 or the other way round
- Therefore, the risk of losing might outweigh the benefit of winning even though both are equally probable

Decision Making Under Uncertainty (with Risk)

- The agent's attitude towards risk can be categorized into the following three types:
  - ***Risk prone:*** In this case, agent prefers high-risk high-return strategy rather than low-risk low-return strategy.
  - ***Risk averse:*** In this case, agent prefers low-risk low-return strategy rather than high-risk high-return strategy.
  - ***Risk neutral:*** If expected value is the same neither can be selected.

Decision Making Under Uncertainty (with Risk)

- ***Maximin return*** (assume the worst state of nature: Select alternative that will maximize the minimum payoff)
- ***Optimism-pessimism index*** (optimism: select alternative that will maximize the maximum payoff; pesimism: Select alternative that will maximize the minimum payoff)
- ***Minimax regret*** (don’t want to regret too much: select alternative that will minimize the maximum regret)
- ***Laplace’s principle of insufficient reason*** (assume equal likely states of nature: select alternative with best average payoff)
- Etc.

Decision Making Under Uncertainty

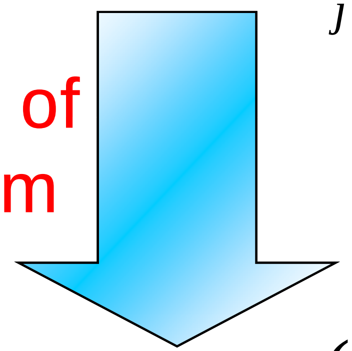
- What if we don't know the probabilities?
- Can belief be evaluated without using the probabilities?
- According to cognitive psychology, when probability distribution is not known, people evaluate belief based on
  - **Degree of comfort** (i.e., selecting the one that needs the least effort)
  - **Degree of optimism** (i.e., selecting the one that we think is the most "fit" given a fitness criteria)
- How to quantify degree of optimism/comfort?

Decision Making Under Uncertainty

- Using Ordered Weighted Averaging (OWA) method [Yager]
- Using OWA we can order the choices according to the best thing happening first, i.e., the best choice has the highest weight
- Use the weights as pseudo-probabilities

$$\vec{W} = [w_1,\ldots,w_n]$$

$$F(a_1,\ldots,a_n) = \sum_j w_j b_j$$



Degree of optimism

$$Opt(W) = \sum_j w_j \frac{(n-j)}{(n-1)}$$

Agent Decision Making

Conclusion

- Decision making is more than following an algorithmic path ← unlike ordinary programs
- Decision making is also more than reasoning ← unlike expert systems
- A library of decision making mechanisms is needed to facilitate implementation of the decision making behavior of software agents

UNIVERSITY OF
CALGARY

Agent-SE:
What will be next?

Will Agent-SE Survive?

Two reasons for survival of Agent-SE:

- **As a next step in evolution of SE**
  - Agent-SE is a natural next step in the evolution of a whole range of software engineering paradigms, such as object-orientation, design patterns and component-ware.
- **Appropriateness for open, networked and distributed systems**
  - Agent-based approach is ideal for developing software for open, networked (i.e., Internet) and distributed systems.

Next Step in Evolutional SE /1

Will Agent SE approach replace OO approach?

No. Agent SE enhances OO and benefits from it.

Why?

- Similarities:
  - Both accept the principle of encapsulation and information hiding and both recognize the importance of interactions.
- Differences:
  - Object-oriented approaches provide minimal support for specifying and managing organizational relationships, i.e., relationships are merely defined by static inheritance hierarchies.
  - Objects encapsulate state and behavior realization they do not encapsulate action choice.

Next Step in Evolutional SE /2

Will Agent-based approach replace contemporary software engineering techniques such as design patterns or component-ware?

No.

Why?

- Agent-based approach provides a higher level of computational abstraction
- This may, in turn, be realized through object-oriented systems or in a design pattern or component-based fashion.

Appropriateness for Open Systems

- **Characteristics of open, networked systems**
  - Control is distributed
  - Constant change is present
- In an open, networked environment, the software model should be based on synthesis or construction, rather than decomposition or reduction [Jennings].

Agent-SE accounts for distributed control and synthesis

What Shall I Do Next?

- Agents seem to be quite interesting software entities. So from now on should I start developing software using agents only?
- No. not necessarily.
- Why?

Why?

Depends on problem you have at hand to be solved.

Software Agents ARE NOT appropriate for

- **System control issues:**
  - Agent based solution may not be appropriate for systems and domains with global constraints (where global control is a must), real-time response required, and where emergent behavior (e.g. deadlocks) have to be avoided.
- **Global perspective & Optimality issues:**
  - All decisions are made based on local knowledge and reaching globally optimal performance is almost impossible.

Why? (cont'd)

Depends on problem you have at hand to be solved.

Agents MAY NOT be appropriate for

- **Performance & Security issues:**
  - All patterns and all outcomes of the interactions are inherently unpredictable. Predicting the behavior of the overall system based on its constituent components is extremely difficult (sometimes impossible).
- **Trust & delegation issues:**
  - Agents should be trusted to delegate tasks to them.
  - Agents should trust each other.

Why? (cont’d)

Opposed to OOAD

- Agent development methodologies are not matured enough
- Development processes are not standardized yet
- Testing agent based systems has yet to be formalized
- Scaling and performance issues still exist
- Libraries of reusable agent tasks are yet to be developed

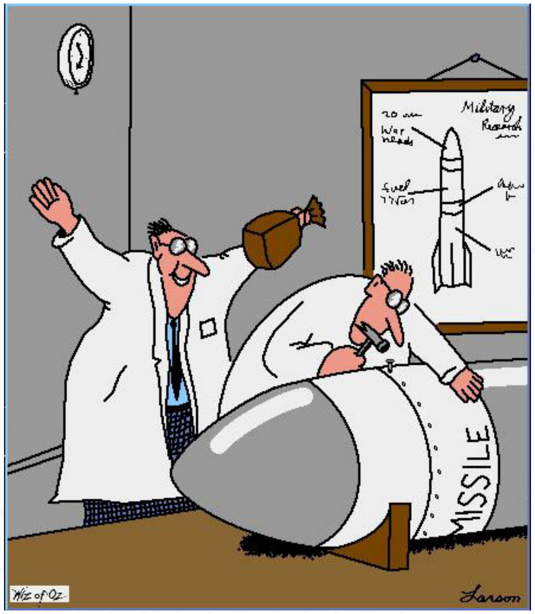
Issues to be Considered

- **Knowledge sharing issues:**
  - Ontology
- **Uncertainty issues:**
  - Uncertain about the parameters of the environment, actions of the other agents, reasoning of the other agents, imperfectly informed of the events happened, etc.
- **Development issues:**
  - Development; testing; deployment

Research Areas

Agent Technology

- Social aspects
  - Organization
  - Emergent behaviour
- Technical aspects
  - Experimental science
  - Artificial science (protocols, individual and group models regression and aggregation models, etc.)

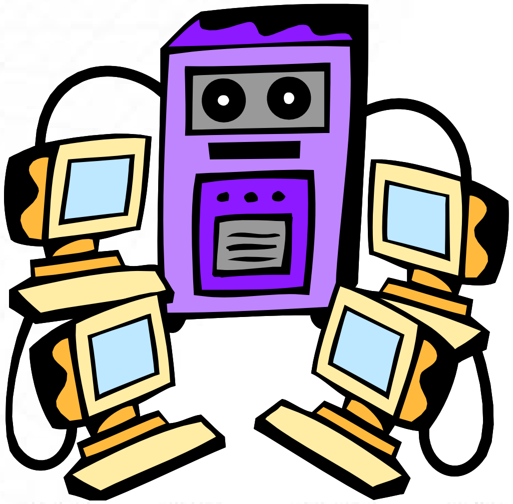


Areas of Investigation

- Methodologies for agent-based analysis and design
- Agent communication
- Knowledge sharing
- Agent-based system architecture and organization
- Metrics for agent-based systems (predictive measures, quality, performance, etc.)
- Agent based testing
- Agent infrastructure, APIs, learning, self-organizing systems, etc.

Conclusions

- Agents are Software++   not alternatives
- Interaction and decision making are at the core of agent system research
- Much must be done on agent-based software development



Questions?

GET ALL THE INFORMATION YOU CAN, WE'LL THINK OF A USE FOR IT LATER.